



# Report Builder in LMS 4.0

**Paul Holden**

Senior Developer, Moodle HQ

**David Matamoros**

Senior Developer, Moodle HQ

**Mikel Martin**

Developer, Moodle HQ

MootIT 2021

























# Introduction

- Report Builder was initially developed for **Moodle Workplace**.
- **Report Builder** consists of **System Reports** and **Custom Reports**.
- We converted two existing reports as an example for others:
  - Task logs
  - Config changes
- We converted one more for this presentation:
  - Cohorts (MDL-73141)

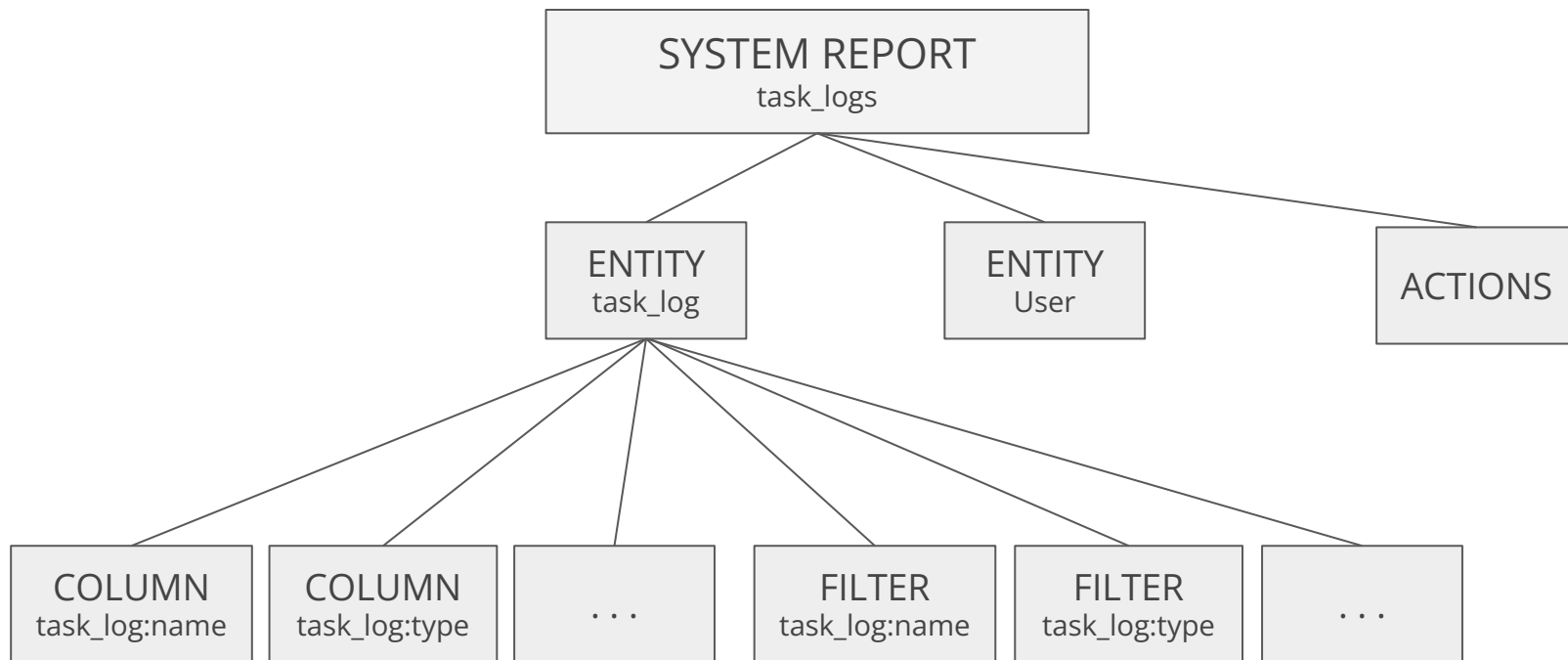
# Cohorts system report

## System

[System cohorts](#)[All cohorts](#)[Add new cohort](#)[Upload cohorts](#)[Filters](#)

Category	Name ^	Cohort ID	Description	Cohort size	Source	
System	Cohort 2 		Description for Cohort 2 😊	0	Created manually	   
System	Cohort 3 	sc2 		1	Created manually	   
Category1	Cohort in Cat1 	cat1 		1	Created manually	   
System	Lionel fans 	123 		2	Created manually	   

# Parts of a system report



# Report Builder

## Entities



# Entities

- An **Entity** is a collection of elements where an element is a **column**, a **filter** or **condition**.
- An entity is re-usable across many reports.
- We can add only the entity elements that we need to each report.

# Report Builder

**Entity: Required  
methods**



# Entities

To create a new entity our class **must extend the base entity class** and include these methods:

`get_default_table_aliases()`

`get_default_entity_title()`

`get_default_entity_name()`

`initialise()`

`get_all_columns()`

`get_all_filters()`



# Entities

## get\_default\_table\_aliases()

Defines the SQL alias for the database tables the entity uses

```
/**
 * Database tables that this entity uses and their default aliases
 *
 * @return array
 */
protected function get_default_table_aliases(): array {
    return ['cohort' => 'c'];
}
```

# Entities

## get\_default\_entity\_title()

Defines the default title for this entity

```
/**
 * The default title for this entity
 *
 * @return lang_string
 */
protected function get_default_entity_title(): lang_string {
    return new lang_string( identifier: 'cohort', component: 'core_cohort');
}
```

# Entities

## get\_default\_entity\_name()

Defines the default internal name for this entity that will be used to manage columns and filters.

```
/**
 * The default machine-readable name for this entity that
 * will be used in the internal names of the columns/filters
 *
 * @return string
 */
protected function get_default_entity_name(): string {
    return 'task_log';
}
```

# Entities

## initialise()

This is where we add the entity elements to our report.

```
/**
 * Initialise the entity
 *
 * @return base
 */
public function initialise(): base {
    $columns = $this->get_all_columns();
    foreach ($columns as $column) {
        $this->add_column($column);
    }

    // All the filters defined by the entity can also be used as conditions.
    $filters = $this->get_all_filters();
    foreach ($filters as $filter) {
        $this
            ->add_filter($filter)
            ->add_condition($filter);
    }

    return $this;
}
```

# Entities

## get\_all\_columns() and get\_all\_filters()

This is where we define the entity columns and filters.

```
/**
 * Returns list of all available columns
 *
 * @return column[]
 */
protected function get_all_columns(): array {
    $tablealias = $this->get_table_alias( tablename: 'cohort');

    // Category/context column.
    $columns[] = (new column(
        name: 'context',
        new lang_string( identifier: 'category'),
        $this->get_entity_name()
    ))
    ->add_joins($this->get_joins())
    ->set_type( type: column::TYPE_INTEGER)
    ->add_fields( sql: "{$tablealias}.contextid")
    ->set_is_sortable( issortable: true)
    ->add_callback(static function(int $contextid): string {
        return context::instance_by_id($contextid)->get_context
    });
}
```

```
/**
 * Return list of all available filters
 *
 * @return filter[]
 */
protected function get_all_filters(): array {
    $tablealias = $this->get_table_alias( tablename: 'cohort');

    // Context filter.
    $filters[] = (new filter(
        filterclass: select::class,
        name: 'context',
        new lang_string( identifier: 'category'),
        $this->get_entity_name(),
        fieldsql: "{$tablealias}.contextid"
    ))
    ->add_joins($this->get_joins())
    ->set_options_callback(static function(): array {
        global $DB;
    });
}
```

# Report Builder

Entity: Columns



# Entities: Columns

- How to create a column:

```
// Category/context column.
$columns[] = (new column(
    name: 'context',
    new lang_string( identifier: 'category'),
    $this->get_entity_name()
))
->add_joins($this->get_joins())
->set_type( type: column::TYPE_INTEGER)
->add_fields( sql: "{$tablealias}.contextid")
->set_is_sortable( issortable: true)
->add_callback(static function(int $contextid): string {
    return context::instance_by_id($contextid)->get_context_name( withprefix: false);
});
```

# Entities: Columns

## Types of columns

- Text
- Integer (Integer numbers)
- Float (Decimal numbers)
- Timestamp (Dates)
- Boolean (Yes / No values)
- Longtext



# Entities: Columns

## Callbacks to format data

- It is easy to format the data received from the database in each column using the **add\_callback()** method.
- We can add all the fields we need using the **add\_field()** or **add\_fields()** methods.

```
// Database column.
$columns[] = (new column(
    name: 'database',
    new lang_string( identifier: 'task_dbstats', component: 'admin'),
    $this->get_entity_name()
))
->add_joins($this->get_joins())
->set_type( type: column::TYPE_TEXT)
->add_fields( sql: "{$tablealias}.dbreads, {$tablealias}.dbwrites")
->set_is_sortable( issortable: true)

->add_callback(static function(string $value, \stdClass $row): string {
    $output = '';
    $output .= \html_writer::div(get_string( identifier: 'task_stats:dbreads', component: 'admin', $row->dbreads));
    $output .= \html_writer::div(get_string( identifier: 'task_stats:dbwrites', component: 'admin', $row->dbwrites));
    return $output;
});
```

# Entities: Columns

- We can use the methods **add\_join()** and **add\_joins()** to add additional SQL JOIN clauses to each column if needed
- **add\_join()** requires a SQL JOIN as a parameter
- **add\_joins()** requires an array of SQL JOINS as a parameter

```
$column->add_join("LEFT JOIN {user} {$usertablealias} ON {$usertablealias}.id = {$maintablealias}.userid");
```

# Entities: Columns

- Sorting

**set\_is\_sortable()** is used to enable/disable sorting on a column. For example we don't need sorting if the column just shows a picture. Note that sorting of LONGTEXT columns cannot be sorted on Oracle.

- Availability

**set\_is\_available()** is used to show/hide an individual column. For example you might want to show a column just for specific users.

# Report Builder

Entity: Filters



# Entities: Filters

1 2 3 4 5 6 7 8 9 10 ... 12 »


Type	User	Start time ▼	Duration	Host
Ad hoc	Admin User	Friday, 16 April 2021, 9:37 AM	0.98 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0.01 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0.01 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0 secs	ubuntu


Filters

Name  
Is any value ▼



Result  
Is equal to ▼ Success ▼

Start time  
Range ▼

Date from  
27 ▼ March ▼ 2021 ▼  ☒ Enable

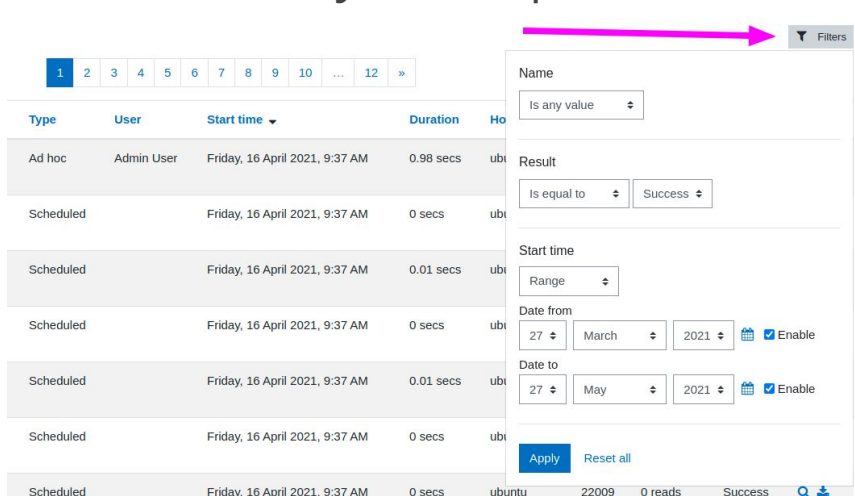
Date to  
27 ▼ May ▼ 2021 ▼  ☒ Enable

Apply Reset all

ubuntu 22009 0 reads Success  

# Entities: Filters

- Filters provide a form for collecting user input, and then return appropriate SQL fragments based on these values.
- They can be accessed clicking on the Filters button located on the top right corner of the system report.



The screenshot shows a Moodle system report table with columns: Type, User, Start time, Duration, and Host. A pink arrow points to the 'Filters' button in the top right corner. The 'Filters' dialog is open, showing fields for Name, Result, Start time, Date from, and Date to, each with a dropdown menu. The 'Apply' button is highlighted in blue.

Type	User	Start time	Duration	Host
Ad hoc	Admin User	Friday, 16 April 2021, 9:37 AM	0.98 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0.01 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0.01 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0 secs	ubuntu
Scheduled		Friday, 16 April 2021, 9:37 AM	0 secs	ubuntu

Filters dialog:

- Name: Is any value
- Result: Is equal to, Success
- Start time: Range
- Date from: 27, March, 2021, Enable
- Date to: 27, May, 2021, Enable
- Buttons: Apply, Reset all

Report summary: 22009, 0 reads, Success

# Entities: Filters

- To create a filter we need to create a new instance of the filter class with these arguments:

1. Filter type class
2. Internal name
3. Visible name
4. Entity name
5. SQL field

And add any custom SQL joins we might need.

```
// Time created filter.
$filters[] = (new filter(
    filterclass: date::class,
    name: 'timecreated',
    new lang_string( identifier: 'timecreated',
    $this->get_entity_name(),
    fieldsql: "{$tablealias}.timecreated"
))
->add_joins($this->get_joins());
```

# Entities: Filters

## Types of filters

- Text
- Date
- Number
- Boolean Select
- Select
- User
- Course selector



# Entities: Filters

Possible values:

- Is any value
- Contains
- Does not contain
- Is equal to
- Is not equal to
- Starts with
- Ends with
- Is empty
- Is not empty

## Text filter

Name

Starts with	▼	Moodle
-------------	---	--------

# Entities: Filters

Possible values:

- Is any value
- Is empty
- Is not empty
- Relative to now
- Range

## Date filter

Start time

Range ⇅

Date from

27 ⇅

March ⇅

2021 ⇅



Enable

Date to

27 ⇅

May ⇅

2021 ⇅



Enable

# Entities: Filters

## Number filter

Possible values:

- Is any value
- Is empty
- Is not empty
- Less than
- Greater than
- Equal to
- Equal or less than
- Equal or greater than
- Range

Street number

Greater than	↕	20
--------------	---	----

# Entities: Filters

## Boolean select filter

Possible values:

- Is any value
- Yes (Checked)
- No (Not checked)

Course visibility

# Entities: Filters

## Select filter

Possible values:

- Is any value
- Equal to
- Not equal to

Result

Is equal to	Success
-------------	---------

We can set our custom options on the dropdown element using the method **set\_options()**

```
->set_options([
    self::SUCCESS => get_string( identifier: 'success'),
    self::FAILED => get_string( identifier: 'task_result:failed', component: 'admin'),
]);
```

# Entities: Filters

## Course selector filter

It allows to select one or more courses selecting them in a dropdown selector

Courses

× Test course: S

× Test course #2

Search

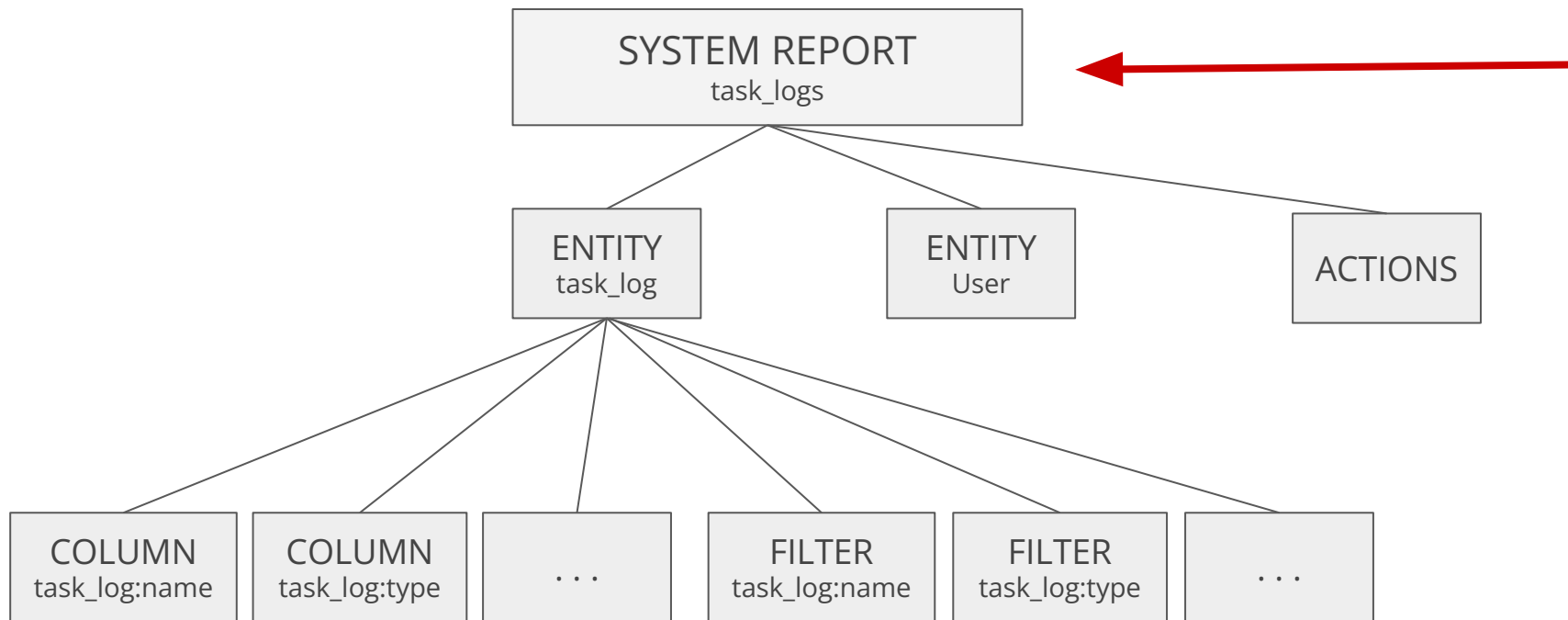


# Report Builder

System report



# Parts of a system report





# System report

To create a new system report **our class must extend the `system_report` class** and include these methods:

```
class task_logs extends system_report {
```

```
    initialise()  
    can_view()  
    get_name()  
    add_columns_from_entities()  
    add_filters_from_entities()  
    add_actions()
```

# System report

## initialise()

- This method is where we add our entities using SQL joins and specify which columns and filters we want to use from them.

```
protected function initialise(): void {  
    // Our main entity, it contains all of the column definitions that we need.  
    $entitymain = new task_log();  
    $entitymainalias = $entitymain->get_table_alias( tablename: 'task_log');  
  
    $this->set_main_table( tablename: 'task_log', $entitymainalias);  
    $this->add_entity($entitymain);  
  
    // Any columns required by actions should be defined here to ensure they're always available.  
    $this->add_base_fields( sql: "{$entitymainalias}.id");  
  
    // We can join the "user" entity to our "main" entity and use the fullname column from the user entity.  
    $entityuser = new user();  
    $entityuseralias = $entityuser->get_table_alias( tablename: 'user');  
    $this->add_entity($entityuser->add_join(  
        join: "LEFT JOIN {user} {$entityuseralias} ON {$entityuseralias}.id = {$entitymainalias}.userid"  
    ));  
}
```

# System report

## initialise()

- We also specify which columns and filters we want to use in our report by calling **add\_columns\_from\_entities()** and **add\_filters\_from\_entities()**
- Columns and filters must be added as **entity\_name:column\_name**

```
$columns = [  
    'task_log:name',  
    'task_log:type',  
    'user:fullname',  
    'task_log:starttime',  
    'task_log:duration',  
    'task_log:hostname',  
    'task_log:pid',  
    'task_log:database',  
    'task_log:result',  
];  
  
$this->add_columns_from_entities($columns);
```

```
$filters = [  
    'task_log:name',  
    'task_log:result',  
    'task_log:timestart',  
];  
  
$this->add_filters_from_entities($filters);
```

# System report

## initialise()

- It is also possible to add custom columns and filters directly from the system report using **add\_column()** and **add\_filter()**

```
// Cohort size column using a custom SQL query to count cohort members.
$cm = database::generate_param_name();
$sql = "(SELECT count($cm.id) as memberscount
        FROM {cohort_members} $cm
        WHERE $cm.cohortid = {$entitymainalias}.id)";
$this->add_column(new column(
    name: 'memberscount',
    new lang_string( identifier: 'memberscount', component: 'cohort'),
    $this->entitymain->get_entity_name()
))
->set_type( type: column::TYPE_INTEGER)
->set_is_sortable( issortable: true)
->add_field($sql, alias: 'memberscount');
```

# System report

## can\_view()

- This method is used to validate if the user has permission to access the report.

```
/**
 * Validates access to view this report
 *
 * @return bool
 */
protected function can_view(): bool {
    $contextid = $this->get_parameter( param: 'contextid', default: 1, type: PARAM_INT);
    $context = context::instance by id($contextid);

    return has_any_capability(['moodle/cohort:manage', 'moodle/cohort:view'], $context);
}
```

# System report

## get\_name()

- This method is where we set the visible name of the system report.

```
/**  
 * Get the visible name of the report  
 *  
 * @return string  
 */  
public static function get_name(): string {  
    return get_string( identifier: 'cohorts', component: 'core_cohort');  
}
```

# System report

## add\_columns\_from\_entities()

```
$columns = [  
    'task_log:name',  
    'task_log:type',  
    'user:fullname',  
    'task_log:starttime',  
    'task_log:duration',  
    'task_log:hostname',  
    'task_log:pid',  
    'task_log:database',  
    'task_log:result',  
];  
  
$this->add_columns_from_entities($columns);
```

- This method adds the columns we define in an array.
- Due to the fact that we can use columns from different entities in the same system report, we need to add each one as

**entity\_name:column\_name**

# System report

## add\_filters\_from\_entities()

- This method adds the filters we define in an array.
- Due to the fact that we can use filters from different entities in the same system report, we need to add each one as **entity\_name:filter\_name**

```
protected function add_filters(): void {  
    $filters = [  
        'cohort:name',  
        'cohort:idnumber',  
    ];  
    $this->add_filters_from_entities($filters);  
}
```






# System report

## add\_actions()

- System reports can have an additional column to add some specific actions to each row.
- Any field needed by the actions can be Passed using add\_base\_fields():

```
$this->add_base_fields("{entitytablealias}.id");
```

Host name	PID	Database	Result	
ubuntu	22009	622 reads 5 writes	Success	 
ubuntu	22009	1 reads 0 writes	Success	 
ubuntu	22009	1 reads 0 writes	Success	 
ubuntu	22009	3 reads 0 writes	Success	 

# System report

## add\_actions()

```
// Edit action. It will be only shown if user has 'moodle/cohort:manage' capability.
$url = new moodle_url( url: '/cohort/edit.php', ['id' => ':id', 'returnurl' => $returnurl]);
$icon = new pix_icon( pix: 't/edit', get_string( identifier: 'edit' ));
$this->add_action((new action($url, $icon))
    ->add_callback(static function(stdClass $row): bool {
        return has_capability( capability: 'moodle/cohort:manage', context::instance_by_id($row->contextid));
    }));

// Delete action. It will be only shown if user has 'moodle/cohort:manage' capability.
$url = new moodle_url( url: '/cohort/edit.php', ['id' => ':id', 'delete' => 1, 'returnurl' => $returnurl]);
$icon = new pix_icon( pix: 't/delete', get_string( identifier: 'delete' ));
$this->add_action((new action($url, $icon))
    ->add_callback(static function(stdClass $row): bool {
        return has_capability( capability: 'moodle/cohort:manage', context::instance_by_id($row->contextid));
    }));

// Assign members to cohort action. It will be only shown if user has 'moodle/cohort:assign' capability.
$url = new moodle_url( url: '/cohort/assign.php', ['id' => ':id', 'returnurl' => $returnurl]);
$icon = new pix_icon( pix: 'i/users', get_string( identifier: 'assign', component: 'core_cohort' ));
$this->add_action((new action($url, $icon))
    ->add_callback(static function(stdClass $row): bool {
        return has_capability( capability: 'moodle/cohort:assign', context::instance_by_id($row->contextid));
    }));
```

# System report

- It is possible to define whether a system report can be downloaded or not using the method **set\_downloadable()** and passing true or false.

```
$this->set_downloadable(true, "Users export");
```

- We can define an initial sorted column on the report by calling **set\_initial\_sort\_column()** and passing the column and the sort order:

```
$this->set_initial_sort_column('task_log:starttime', SORT_DESC);
```

# System report

- The **display name** of a column can be overridden in case we want to specify our own:

```
$column->set_title(new lang_string('user', 'admin'));
```

# System report

- We can add the same entity more than once to a system report using **set\_entity\_name()** and **set\_table\_alias()**

```
// We can join the "user" entity to our "main" entity using standard SQL JOIN.
$entityuser = new user_entity();
$entityuseralias = $entityuser->get_table_alias( tablename: 'user');
$this->add_entity($entityuser
    ->add_join( join: "LEFT JOIN {user} {$entityuseralias} ON {$entityuseralias}.id = {$logentityalias}.userid"
    );

// We can join again the "user" entity if we need to add another SQL JOIN. We just need to specify a different
// entity name and a different table alias. After that, remember to use this new name to add the columns we want.
$entityuser2 = new user_entity();
$entityuseralias2 = 'relateduser';
$this->add_entity($entityuser2
    ->add_join( join: "LEFT JOIN {user} {$entityuseralias2} ON {$entityuseralias2}.id = {$logentityalias}.relateduserid"
    ->set_entity_name( entityname: 'relateduser')
    ->set_table_alias( tablename: 'user', alias: 'relateduser')
    );
```

# Report Builder

Custom reports



# Custom Reports

- The Custom Reports **Editor** is a flexible tool to create reports using an interactive **drag & drop interface** to define the report content
- To create a new report you just need to select a “**Report source**”
- Available sources at this moment are: **Cohorts, Users** and **Courses**

# Custom Reports

# Editor

## Users report

[Edit details](#)[Close](#)[Editor](#) [Audience](#) [Access](#)[Preview](#)

User	
Full name	+
Full name with link	+
Full name with picture	+
Full name with picture and link	+
User picture	+
First name	+
Surname	+
Email address	+
City/town	+
Country	+
First name - phonetic	+
Surname - phonetic	+
Middle name	+
Alternate name	+
ID number	+

+ USER • FULL NAME WITH PICTURE	+ USER • USERNAME	+ USER • EMAIL ADDRESS
Name	Username	Email address
No aggregation	No aggregation	No aggregation
LR Lionel Richie	user05	lionel@local.host
EI Estudiante Ingenieria	user03	user03@local.host
PI Profesora Ingenieria	user02	user02@local.host
U0 User 01	user01	user01@local.host
MJ Michael Johnson	tool_generator_000100	tool_generator_000100@example.com
MM Мария Морозова	tool_generator_000099	tool_generator_000099@example.com
MJ Michael Johnson	tool_generator_000098	tool_generator_000098@example.com
伟张 伟张	tool_generator_000097	tool_generator_000097@example.com
LH Leonie Hoffmann	tool_generator_000096	tool_generator_000096@example.com
SD Sophia Davis	tool_generator_000095	tool_generator_000095@example.com
LM Lukas Meyer	tool_generator_000094	tool_generator_000094@example.com

[Conditions](#)

+ USER • FULL NAME

Does not contain

+ USER • SURNAME

Contains

[Apply](#)[Reset all](#)[Filters](#)

+ USER • FULL NAME

Full name

+ USER • EMAIL ADDRESS

Email address

+ USER • DEPARTMENT

Department

[Sorting](#)[+](#) [|](#) [\[icon\]](#) [Username](#)



# Custom Reports

## Editor

- To add new **columns** to the report we just need to select from the list of available columns on the left side
- We can select which **filters** we want to use in the report and also set **conditions**.
- We can also enable **sorting** and set the default sort direction for each column
- We can also choose the report **card view** settings for defining how the report will be displayed on narrow devices

## Difference between Filters and Conditions

- **Conditions** are a predefined set of criteria that **are applied** when viewing a report. Conditions cannot be changed in viewing mode.
- **Filters** are a predefined set of criteria that **are not applied automatically but are available** for report viewers (via the Filters icon). Viewers of the report then have the ability to reduce the amount of data further via filters.

# Custom Reports

## Audience

- Report audiences indicate which users have access to the report. They can also be used as recipients in **scheduled reports**.
- We can define an audience for the report selecting All users, individual users, users with assigned system roles or members of cohorts.

# Custom Reports


## Audience


- You can create the following audiences:
  - **All users:** Use this audience to give access to all users in the site to any report
  - **Assign system roles:** Select at least one system role
  - **Manually added users:** Select at least one user (via name or e-mail address)
  - **Member of cohort:** Select at least one cohort


# Custom Reports


# Audience


[Editor](#) [Audience](#) [Schedules](#) [Access](#)


Search 


Site 

All users 

Assigned system role 


Manually added users 

Member of cohort 


Member of cohort 

New system cohort

OR

Human resources 

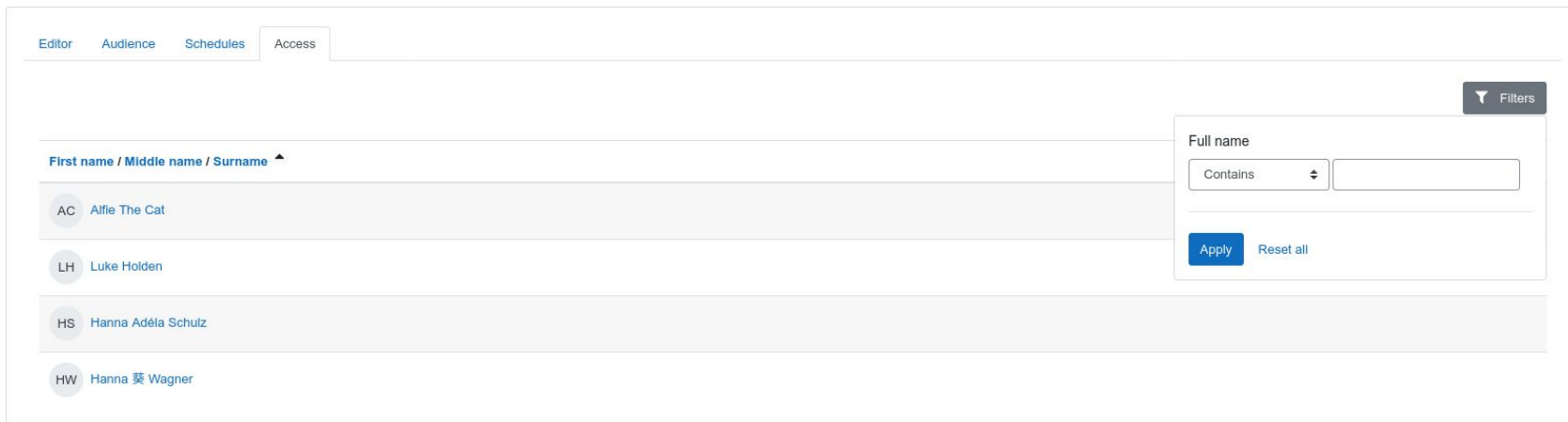
Hanna Adéla Schulz, Hanna 葵 Wagner



# Custom Reports

## Access

- Access tab shows the list of users that can access this report. These users have been set in the Audience.



The screenshot displays the 'Access' tab of a Moodle Custom Report. At the top, there are four tabs: 'Editor', 'Audience', 'Schedules', and 'Access', with 'Access' being the active tab. Below the tabs, a header row reads 'First name / Middle name / Surname' with a small upward arrow. A list of four users is shown, each with a circular icon containing initials and their full name: 'AC Alfie The Cat', 'LH Luke Holden', 'HS Hanna Adéla Schulz', and 'HW Hanna 葵 Wagner'. On the right side, there is a 'Filters' sidebar. It contains a 'Full name' section with a 'Contains' dropdown menu and an empty text input field. Below the input field are two buttons: 'Apply' (in blue) and 'Reset all' (in grey).

First name / Middle name / Surname
AC Alfie The Cat
LH Luke Holden
HS Hanna Adéla Schulz
HW Hanna 葵 Wagner

Filters

Full name

Contains

Apply Reset all

## Custom Reports

## Schedules

- The report scheduler lets you configure the automatic delivery of reports to specific audiences
- We can set the **file format, date** when it should be sent, **recurrence** and **how the data will be viewed**
- We can set an **audience** for the scheduled report
- We can set any **custom email message**

# Schedules





One more  
thing...



# Where to find the code?

All the Report Builder related issues can be found in this Epic:

<https://tracker.moodle.org/browse/MDL-70343>

It contains the System Report, Custom Reports and all follow-up issues related to Report Builder development.

## Documentation

[https://docs.moodle.org/dev/index.php?title=Report\\_builder\\_API](https://docs.moodle.org/dev/index.php?title=Report_builder_API)



